## General Disclaimer

## One or more of the Following Statements may affect this Document

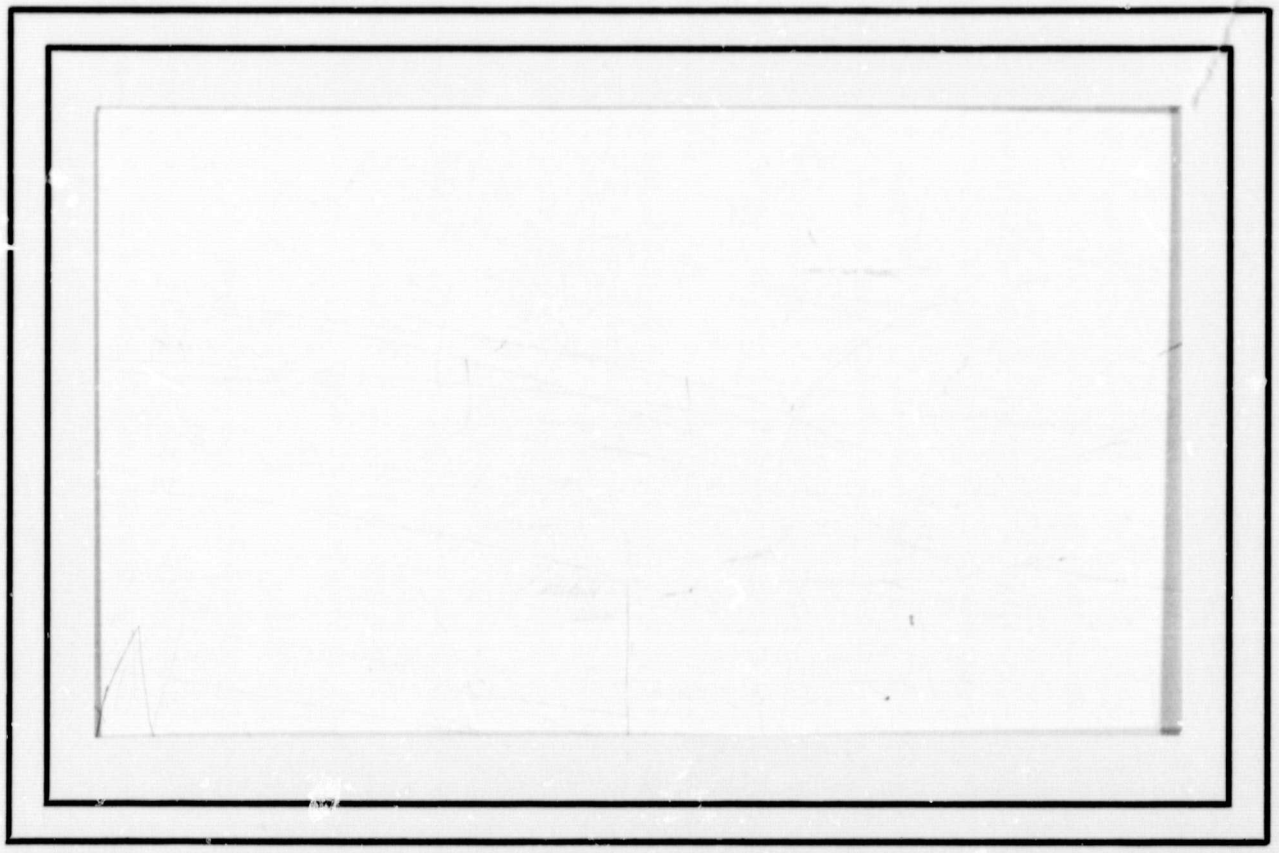
- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.




# UNIVERSITY OF MARYLAND
# COMPUTER SCIENCE CENTER

**COLLEGE PARK, MARYLAND**

Technical Report TR-165 August 1971
GJ-1067 and NGL-21-002-008

# GRAAL - A Graph Algorithmic Language

by

Werner C. Rheinboldt
Victor R. Basili
Charles K. Mesztenyi

Lecture to be presented at the Second
IBM Research Symposium on Sparse Matrices
and Their Applications, Sept. 9-10, 1971.
To be published in the Proceedings of that
Symposium by Plenum Press, New York.

This research was supported in part by Grant GJ-1067 from the National Science Foundation and by Grant NGL-21-002-008 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

## Abstract

In an earlier report (University of Maryland, Computer Science Center, TR-158, June 1971) the authors presented a new algorithmic language, GRAAL, for describing and implementing graph algorithms as they arise in applications. In its original form GRAAL was defined as an extension of ALGOL 60 (Revised). In this paper, a FORTRAN-based version of the language, called FGRAAL, is discussed. The paper begins with a brief review of the set theoretical foundation of the basic language; then the FGRAAL grammar is summarized, followed by five examples of FGRAAL subroutines. One of these represents a new algorithm for finding strong components of a digraph.

# GRAAL - A Graph Algorithmic Language[1)]

Werner C. Rheinboldt, Victor R. Basili, Charles K. Mesztenyi

Computer Science Center, University of Maryland

## 1. Introduction

The authors recently developed a new algorithmic language, GRAAL, for describing and implementing graph algorithms as they arise in applications (see Rheinboldt, Basili, Mesztenyi (1971A)). These algorithms involve a wide variety of graphs of different types and complexity, such as highly structured graphs with multiple arcs and self loops and with functions defined over the nodes and arcs, or very large, sparse graphs in which only the adjacency relations between the nodes are of interest. One of the design objectives of GRAAL was to allow for this wide range of possibilities without degrading overly the efficient implementation and execution of any specific class of algorithms. A second objective relates to the general need for a language which facilitates the design and communication of graph algorithms independent of a computer. In line with this, the aim was to ensure a concise and clear description of such algorithms in terms of data objects and operations natural to graph theory and as free as possible from strictly technical programming requirements.

During recent years several graph algorithmic languages have been proposed, and a brief survey of these efforts can be found in the authors' cited article. The objectives of these languages and their design characteristics differ largely from those of GRAAL.

The design of GRAAL has been based on a set algebraic model of graph theory which defines the graph structure in terms of

[1)] This work was supported in part by NSF Grant GJ-1067 and NASA Grant NGL-21-002-008.

morphisms between certain set algebraic structures over the node set and arc set. GRAAL is a modular language in the sense that the user specifies which of these mappings are available for any graph. This allows for considerable flexibility in the selection of the storage representation for different graph structures and is one of the means of meeting the first objective of the language.

In its original form GRAAL has been defined as an extension of ALGOL 60 (Revised). In this article we present a FORTRAN-based version of the language, called FGRAAL. Implementation of this extension of FORTRAN is presently under way. In Section 2 below we discuss briefly the set theoretic foundation of GRAAL, which also constitutes a definition of the semantics of the principal graph operations under the language. Then in Section 3 the grammar of FGRAAL is summarized and, finally, Section 4 presents several examples of FGRAAL subroutines relating generally to the principal topic of this Symposium.

## 2. Set Algebraic Basis of GRAAL

In this section, capital letters X,S, etc., stand for finite sets, and besides the standard set operations of union ($\cup$), intersection ($\cap$), and difference (~), the symmetric sum ($\Delta$) will be used. For any set X the cardinality is denoted by $|X|$, $P(X)$ is its power set, and

$$P_k(X) = \{S \in P(X) \mid |S| = k\},\ k = 0,1,\ldots,|X|.$$

It is well-known that under union, intersection, and complementation (in X), $P(X)$ is a free Boolean algebra with the members of $P_1(X)$ as generators. To obtain a different algebraic structure, let GF(2) be the binary Galois field with the integers 0,1 as elements. Then $P(X)$ becomes a vector space over GF(2) if the symmetric sum is used as addition and the scalar product is defined by $\lambda S = \emptyset$ for $\lambda = 0$, ($\emptyset$ the empty set), and $\lambda S = S$ for $\lambda = 1$. The elements of $P_1(X)$ now form a basis.

For any X,Y we denote by $B(X,Y)$ the class of all morphisms $\psi: P(X) \to P(Y)$ between the Boolean algebras $P(X)$,$P(Y)$. Any $\psi \in B(X,Y)$ is uniquely characterized by the image sets $\psi\{x\} \in P(Y)$ of the generators $\{x\} \in P_1(X)$, and

$$\psi S = \bigcup_{x \in S} \psi\{x\}, \quad \forall S \in P(X).$$

Analogously, we define $L(X,Y)$ as the class of all linear mappings $\psi: P(X) \to P(Y)$ between the vector spaces $P(X)$ and $P(Y)$. As before, any $\psi \in L(X,Y)$ is uniquely determined by the specification of $\psi\{x\} \in P(Y)$ for all basis elements $\{x\} \in P_1(X)$, and we have

$$\Delta S = \Delta_{x \in S} \psi\{x\}, \quad \forall\, S \in P(X).$$

If G is a graph with node set V and arc set A, the elements of P(V) and P(A) constitute the basic data objects for all operations on G under GRAAL, and the structure of G is given by certain Boolean or linear mappings between the two power sets.

We define an undirected pseudograph as a triple $G = (V,A,\Phi)$ consisting of a node set V, an arc set A, and an incidence operator $\Phi$ with the properties

$$\text{(i) } \Phi \in B(A,V), \quad \text{(ii) } \Phi\{a\} \in P_1(V) \cup P_2(V), \quad \forall\, a \in A.$$

We speak of a multigraph if always $|\Phi\{a\}| = 2$, and of a graph if, in addition, the restricted mapping $\Phi: P_1(A) \to P_2(V)$ is injective. The other graph operators, presently available in GRAAL, can then be defined as follows (always with $v \in V$, $a \in A$):

| | |
|---|---|
| star op. | $\sigma \in B(V,A)$, $\sigma\{v\} = \{a \in A \mid v \in \Phi\{a\}\}$, |
| boundary op. | $\partial \in L(A,V)$, $\partial\{a\} = (\lvert\Phi\{a\}\rvert - 1)\Phi\{a\}$, |
| co-boundary op. | $\delta \in L(V,A)$, $\delta\{v\} = \{a \in \sigma\{v\} \mid \lvert\Phi\{a\}\rvert = 2\}$, |
| adjacency op. | $\alpha \in B(V,V)$, $\alpha\{v\} = \{u \in V \mid \exists a \in \sigma\{v\}, \Phi\{a\} = \{u,v\} \in P_2(V)\}$. |

Other operators are possible and may be included later. Each one of these operators can be used in place of $\Phi$ to characterize graphs of a specific type. In particular, pseudographs can be defined in terms of the star operator, multigraphs in terms of $\partial$ or $\delta$, and graphs by the adjacency operator. As an example, we consider only the last part of this statement. Note that $\alpha$ satisfies

$$\text{(i) } \alpha \in B(V,V), \quad \text{(ii) } v \notin \alpha\{v\}, \quad \forall\, v \in V$$
$$\text{(iii) } u \in \alpha\{v\} \text{ if and only if } v \in \alpha\{u\}, \quad \forall\, u,v \in V.$$

Now let V be any set and $\alpha$ any Boolean mapping with these properties. Then $G = (V,A,\Phi)$ with

$$A = \{\{u,v\} \in P_2(V) \mid u \in \alpha\{v\}\},$$
$$\Phi \in B(A,V),\ \Phi\{a\} = \{u,v\} \text{ if } a = \{u,v\}, \quad \forall a \in A$$

is a well-defined graph with $\alpha$ as its adjacency operator. We call $(V,\alpha)$ the node form representation of G.

The development presented so far is easily carried over to directed graphs. A directed pseudograph shall be a quadruple

$G = (V,A,\Phi_+,\Phi_-)$ consisting of a node set V, an arc set A, and a positive and negative incidence operator

$$\Phi_+,\Phi_- \in B(A,V),\ \Phi_+\{a\},\ \Phi_-\{a\} \in P_1(V),\quad \forall a \in A.$$

It is convenient to consider also the combined operator

$$\Phi \in B(A,V),\ \Phi\{a\} = \Phi_+\{a\} \cup \Phi_-\{a\},\quad \forall a \in A$$

which permits the definition of the terms directed multigraph and directed graph (digraph) as in the undirected case. Then as before we can introduce the following graph operators:

| | |
|---|---|
| star op. | $\sigma_+,\sigma_- \in B(V,A),\ \sigma_\pm\{v\} = \{a\in A \mid v\in\Phi_\pm\{a\}\}$ |
| boundary op. | $\partial_+,\partial_- \in L(A,V),\ \partial_\pm\{a\} = \Phi_\pm\{a\}$ |
| co-boundary op. | $\delta_+,\delta_- \in L(V,A),\ \delta_\pm\{v\} = \sigma_\pm\{v\}$ |
| adjacency op. | $\alpha_+,\alpha_- \in B(V,V),\ \alpha_\pm\{v\} = \{u\in V \mid \exists a\in\sigma_\pm\{v\}, u=\Phi_\mp\{a\}\}.$ |

In each case, it is useful to introduce also the combined operators

$$\sigma = \sigma_+ \cup \sigma_-,\ \partial = \partial_+ \Delta \partial_-,\ \delta = \delta_+ \Delta \delta_-,\ \alpha = \alpha_+ \cup \alpha_-.$$

Together, the positive and negative version of any one of these operators allow the characterization of a specific type of graph. In particular, as in our undirected example, we can introduce a node form representation $(V,\alpha_+,\alpha_-)$ of a digraph.

## 3. Summary of FGRAAL

In this section we present a brief overview of FGRAAL, that is, of the GRAAL version defined as an extension of FORTRAN. Rules of FORTRAN not affected by the introduction of FGRAAL are not repeated here.

### A. Declarations

| | |
|---|---|
| Set declaration | SET X,Y,... |
| List declaration | 'type' STAQUE L,K,... |
| Property declaration | 'type' PROPERTY R,T,... |
| Graph declaration | GRAPH G('module'),... |

In FGRAAL, sets constitute a new data type, and 'type' stands either for any one of the existing FORTRAN data types or for SET. An atomic set consists of exactly one element, and any set is either empty or a union of atomic sets. A list is a doubly-open linked list structure which may be used as a stack or a queue. A property may be associated with any atomic set. The property for a particular atomic set exists and may be referenced only after it has been assigned a value; otherwise, a default value is

returned. Graphs represent specific data structures together with certain operations for manipulating them. The graph declaration identifies the type of data structure used and the family of graph operators available with it. The language is modular, that is, only some of the possible graph operators are usable with any specific graph. Four modules are presently defined in the language representing directed and undirected pseudographs, as well as directed and undirected graphs in node form.

B. Set Operators and Functions

Set constant: & or .EMPTY., the empty set.
Set operators, in increasing order of precedence:
.D. difference, .U. union, .S. symmetric sum,
.I. intersection.
Set relational operators: .EQ. equal relation, .NE. not equal relation, .IN. inclusion relation.

| | |
|---|---|
| CREATE(0) | Creates atomic set with next sequence number |
| CREATE(P1: V1,...) | Returns atomic set with matching "property: value" pairs or creates it if nonexistent |
| ATOM(I) | Returns atomic set with sequence number I or empty set if nonexistent |
| ELT(I,S) | Returns atomic set located in ith position within the set S |
| INDEX(A,S) | Returns position number of atomic set A in S |
| SIZE(S) | Cardinality of S |
| PARITY(S) | TRUE or FALSE if cardinality of S is odd or even, respectively |
| COUNT(0) | Maximal sequence number used |
| COUNT(A) | Sequence number of atomic set A |
| SUBSET(X,E) | Returns set of all elements satisfying logical expression E |
| CHECK(P,A) | TRUE if property P has a value for the atomic set A, otherwise FALSE |

Each atomic set carries a sequence number which is assigned to it at the time of its creation. A set is a union of atomic sets ordered in ascending order of their sequence number, thereby allowing for an efficient manipulation of sets. All sequence numbers assigned to atomic sets are retained in an element sequence which serves the dual purpose of cataloging the existing atomic sets and of providing the linkage between atomic sets and the properties assigned to them.

C. List Operators and Functions

List constant: # or .NIL., the empty list.
List operator: L : K : M or L .ET. K .ET. M
Standard concatenation of lists of same type

| | |
|---|---|
| FIRST(L), DFIRST(L) | The first or last element is returned |
| LAST(L), DLAST(L) | and also deleted if D is present. |

D. Graph Functions

| | |
|---|---|
| NODES(G), ARCS(G) | Return node or arc set of G |
| INC, PINC, NINC | Incidence operators |
| STAR, PSTAR, NSTAR | Star operators |
| BD, PBD, NBD | Boundary operators |
| COB, PCOB, NCOB | Co-boundary operators |
| ADJ, PADJ, NADJ | Adjacency operators |

The P-(positive) and N-(negative) operators are only available for directed graphs. For graphs in node form only the adjacency operators are defined. All graph functions are of type SET.

E. Graph Construction Statements

| | |
|---|---|
| ASSIGN G,A | The atomic set A is assigned to G as a node. |
| ASSIGN G,A-B | The atomic sets A,B are assigned to G as a pair of adjacent nodes. |
| ASSIGN G,A-B,C | The atomic sets A,B,C are assigned to G as arc C with nodes A and B. |
| DETACH G | Delete all nodes and arcs from G. |
| DETACH G,S | The elements of the set S are deleted from G. Nodes are deleted together with all incident arcs. |
| DETACH G,S-T | Delete all arcs connecting the sets S and T of nodes of G. |

F. Property Functions

The statement P(A) = E assigns to the atomic set A as value of the property P the value of the expression E. When P(A) is referenced, e.g., as part of an expression, its current value is retrieved.

G. Remove Statement

REMOVE S,T,...,P,Q,...,PP(X),PQ(Y),...

The remove statement may have the following arguments:

(a) Sets: All atomic sets contained in these sets are removed from the universal sequence.

(b) Property names: The property is removed from all atomic sets to which it had been assigned.

(c) Property names followed by a set in parentheses: The property is removed from the atomic sets contained in the specified set.

H. Iterative Statements

```
DO K FORALL X .IN. S
DO K WHILE E
```

Both statements cause the repetitive execution of all subsequent statements including that with label K. In the first case, X is set equal to the atomic sets in S in succession, and in the second, the iteration continues until the logical expression E is FALSE. The DO range is skipped completely if S is empty or E is initially FALSE.

## 4. Examples

The FGRAAL subroutines presented in this section are intended to illustrate some features of the language and to show how the set theoretical structure introduces a certain novelty into the design of graph algorithms. In the interest of clarity, no particular attempt was made to optimize the programs. The choice of algorithms was influenced by the topic of this Symposium. The first two subroutines determine the strong components of a directed graph G in node form, the third constructs the corresponding acyclic condensation graph CG, and the fourth provides a topological sort of the nodes of G. With a maximal transversal algorithm --excluded for space reasons--this completes a set of programs for partitioning a large sparse matrix. Since in these programs G is always in node form, only the adjacency operators are used. As an example involving some other graph operators, this section concludes with a routine for constructing a spanning tree of a pseudograph.

The following subroutine finds the strong component of G defined by the node V and returns the node set of this component in the set X. The approach used here appears to be new; each node is accessed at best twice.

```
      SUBROUTINE STCOMP(G,V,X)
      GRAPH G
      SET V,X,P,PT,N,NT,SP,SN
C  X  NODES OF THE COMPONENT FOUND SO FAR
C  P  POS. REACHABILITY SET: NODES REACHED FROM X
C  N  NEG. REACHABILITY SET: NODES FROM WHICH X IS REACHABLE
C  PT SUBSET OF P FOR WHICH 'PADJ' HAS NOT BEEN COMPUTED
C  NT SUBSET OF N FOR WHICH 'NADJ' HAS NOT BEEN COMPUTED
      X = V
      I = 1
      PT = PADJ(G,V)
      NT = NADJ(G,V)
      P = PT
      N = NT
      DO 50 WHILE (PT .NE. &) .AND. (NT .NE. &)
```

```
C  CHECK FOR INTERSECTION OF REACHABILITY SETS
       SP = PT .I. NT
       IF (SP .NE. &) GO TO 30
C  NO, ENLARGE ALTERNATIVELY THE TWO REACHABILITY SETS
       GO TO (10,20), I
C  POSITIVE REACHABILITY SET
 10    PT = PADJ(G,PT) .D. P
       P = P .U. PT
       I = 2
       GO TO  50
C  NEGATIVE REACHABILITY SET
 20    NT = NADJ(G,NT) .D. N
       N = N .U. NT
       I = 1
       GO TO 50
C  YES, ENLARGE COMPONENT
 30    SN = SP
C  ENLARGE COMPONENT BY TRAVERSING BACK FROM INTERSECTION
       DO 40 WHILE (SP .NE. &) .OR. (SN .NE. &)
       X = X .U. SP .U. SN
       SP = P .I. NADJ(G,SP) .D. X
 40    SN = N .I. PADJ(G,SN) .D. X
C  REDEFINE SETS P, PT, N AND NT
       SP = P .I. X
       SN = N .I. X
       PT = (PT .D. SP) .U. (PADJ(G,SN) .D. (P .U. X))
       NT = (NT .D. SN) .U. (NADJ(G,SP) .D. (N .U. X))
       P = (P .D. SP) .U. PT
       N = (N .D. SN) .U. NT
 50    CONTINUE
       RETURN
       END
```

The previous subroutine can be called repeatedly to find all strong components of G. A more optimal combined program is possible.

```
       SUBROUTINE ALLSTC (G,L)
       GRAPH G
       SET STAQUE L
       SET N,X,V
       L = #
       N = NODES(G)
       DO 10 WHILE N .NE. &
C  GET A NODE
       V = ELT(1,N)
C  GET THE CORRESPONDING COMPONENT
       CALL STCOMP (G,V,X)
       L = L : X
C  SUBTRACT THE NODES FROM STARTING SET
 10    N = N .D. X
```

```
      RETURN
      END
```

From the list L produced here the next program now constructs the acyclic condensation of the graph G.

```
      SUBROUTINE CONDS(G,L,CG,REF)
      GRAPH G, CG
      SET STAQUE L
      SET PROPERTY REF
      SET S,X
C  LOOP FOR THE NODES OF CG
      DO 10 WHILE L .NE. #
C  CREATE NODE
      S = DFIRST(L)
      X = CREATE(REF : S)
      ASSIGN CG,X
      S = ADJ(G,S)
C  LOOP TO CREATE THE ADJACENCY
      DO 10 FORALL  T .IN. L
      IF (S.I.T .NE. &) ASSIGN CG,X - CREATE(REF:T)
 10   CONTINUE
      RETURN
      END
```

Finally, the nodes of CG are sorted into the set array A such that the existence of a directed path from node X = A(I) to node Y = A(J) implies that I < J.

```
      SUBROUTINE TOPSRT (CG,A,M)
      GRAPH CG
      SET A
      DIMENSION A(M)
C  R  SET TO COLLECT THE NODES TO BE SORTED
C  T  SET TO COLLECT THE ALREADY SORTED NODES
      SET R,T,X
      M = 0
      T = &
      R = SUBSET(X,X .IN. NODES (CG) .AND. NADJ(CG,X).EQ.&)
C  LOOP TO PROCESS ELEMENTS OF R
      DO 10 WHILE R .NE. &
      X = ELT(1,R)
C  ARE THE PREDECESSORS OF X ALREADY SORTED
      IF (.NOT. (NADJ(CG,X) .IN. T)) GO TO 10
C  YES, PLACE X IN THE ARRAY A AND T
      M = M + 1
      A(M) = X
      T = T .U. X
C  AND ADD ITS POSITIVE ADJACENCY NODES TO R FOR PROCESSING
      R =  R .U. PADJ(CG,X)
```

```
C  NO, DELETE X FROM R TO BE ADDED BACK LATER
 10     R = R .D. X
        RETURN
        END
```

The following subroutine is intended to show the use of some of the other graph operators. It generates a directed spanning tree TREE with root U for a connected (directed or undirected) pseudograph.

```
        SUBROUTINE SPTREE (G,U,TREE)
        GRAPH G, TREE
        SET U,S,T,X,Y,W,A
C  INITIALIZE TREE AND S WITH U
        ASSIGN TREE, U
        S = U
        T = COB(G,U)
C  LOOP TO PROCESS OUTGOING ARCS FROM NODE SET S
        DO 20 WHILE T .NE. &
        DO 10 FORALL  A .IN. T
C  GET END NODE WHICH IS NOT IN S
        W = BD(G,A)
        Y = W .D. S
C  ADD IT TO S AND ASSIGN IT TO TREE
        IF (Y .EQ. &) GO TO 10
        S = S .U. Y
        X = W .D. Y
        ASSIGN TREE, X-Y, A
 10     CONTINUE
C  GET NEW SET OF OUTGOING ARCS
 20     T = COB(G,S)
        RETURN
        END
```

## 5. References

Rheinboldt, W. C., Basili, V. R., and Mesztenyi, C. K. (1971A). "On a Programming Language for Graph Algorithms", University of Maryland, Computer Science Center, Technical Report TR-158; SIAM Journal on Computing, submitted for publication.